# Exemplos Práticos de Programação Visual em C#

por **Paulo Cortez** e **Hélder Quintela**

Unidade de Ensino
Departamento de Sistemas de Informação
Escola de Engenharia
Universidade do Minho
Guimarães, Portugal

Janeiro, 2008

# Índice

# Índice de Figuras

# Índice de Tabelas

# 1 Introdução

Embora existam livros escritos na língua portuguesa sobre programação em C#, ver por exemplo o livro [Marques e Pedroso, 2005], estes não focam a programação visual. Assim, para comaltar esta falha, os docentes da disciplina de Tecnologia de Computadores I, do Mestrado Integrado em Engenharia Electrónica Industrial e Computadores, ano lectivo de 2007-08, decidiram escrever este texto.

Parte-se do princípio que os leitores já conhecem a linguagem de programação C# (caso contrário convém ler por exemplo o livro acima mencionado). Também se parte do princípio que o leitor já utilizou um ***Graphic User Interface (GUI)*****,** com janelas, ménus, botões e outros componentes. Assim, o objectivo deste texto é ensinar somente programação visual em C#, via o mecanismo de *Windows Forms*, através da apresentação de um conjunto de exemplos práticos. De realçar que todo o código apresentado neste livro foi executado na **plataforma integrada de desenvolvimento (IDE)** do *MS Visual Studio* 2005 e/ou 2008.

A versão mais actual do *Visual Studio* é a 2008, existindo várias alternativas em termos de funcionalidades disponíveis. Para uma utilização de aprendizagem/académica, recomenda-se a versão gratuita (*MS Visual Studio* 2008 *Express*), com *download* disponível no sítio da *Microsoft*.

# 2 Introdução ao *Visual Studio .Net* C#

## 2.1 Criar uma aplicação

**Explicação**

Para a criação de uma aplicação no *Visual Studio* é importante compreender 3 conceitos: **solution**, **project** e **project item**.

Uma solução (**solution**) é um conjunto de projectos e ficheiros relacionados que integrarão a aplicação. A utilização de diversos projectos para uma solução única permite melhorar aspectos como: a edição, o controlo de erros (*debug*) e a execução de todas as partes da aplicação a partir de uma única sessão. Trata-se de um conceito importante quando se lida com aplicações muito complexas e compostas por várias áreas que devem ser desenvolvidas de forma autónoma, por modo a reduzir a complexidade no desenvolvimento, assegurando contudo facilidade de integração posterior.

Um projecto (**project**) consiste num conjunto de **project items** (e.g. formulários, classes, ficheiros XML, relatórios), representando normalmente um componente da aplicação. Quando se desenvolvem aplicações relativamente simples, o conceito de solução confunde-se com o de projecto, uma vez que a solução será composta por um único projecto. No desenvolvimento de uma aplicação, o primeiro passo consiste então na criação de uma **solution**, tratando-se de um processo transparente, uma vez que a solução é criada quando se cria o primeiro projecto para a aplicação.

**Exemplo**

Para criar uma solução, devem segui-se os seguintes passos:

1. Seleccionar **Ficheiro | New | Project**. A janela *New Project* é apresentada (Figura 1).

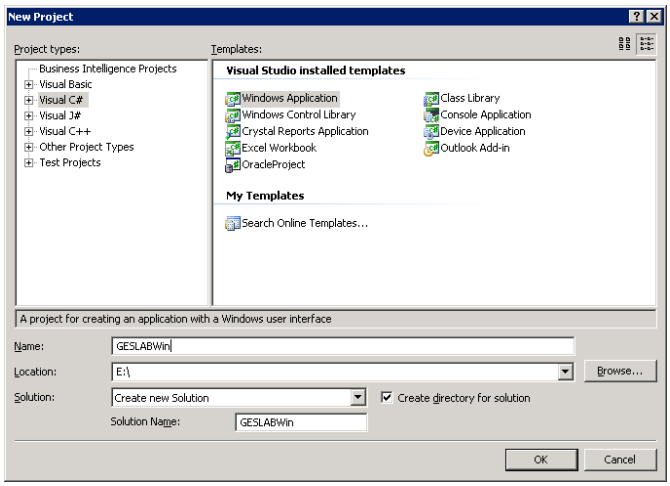


**Figura 1 – Exemplo da criação de um projecto no *Visual Studio***

2. Definir o tipo de projecto a criar (e.g. **Visual C#**), e seleccionar para o tipo de projecto escolhido o *template* desejado (e.g. **Windows Application**, **Console Application**, **Class Library**), de acordo com o tipo de aplicação a desenvolver.
3. Introduzir o nome do projecto, devendo-se ter-se alguns cuidados e usar algumas convenções, como utilizar uma abreviatura do nome da aplicação com um sufixo indicando o tipo de projecto. Por exemplo, no caso de estarmos a desenvolver uma aplicação para gestão de componentes de um laboratório em ambiente *Windows*, poderá dar-se ao projecto o nome GESLABWin.
4. Definir a localização onde serão guardados todos os ficheiros da solução.
5. Clicar em Ok.

Após estes passos, na janela **Solution Explorer** (Figura 2) do *IDE* (Ambiente de Trabalho do *Visual Studio*), é apresentado de forma hierárquica o projecto criado e os seus respectivos items. Por defeito, quando um projecto do tipo *Windows Application* em C# é criado, são adicionados os items **Form1.cs** e **Program.cs**. O **Form1** é um formulário base, que será por defeito (embora se possa alterar) o primeiro a ser visualizado quando a aplicação inicia. Por sua vez, o **Program.cs** é o ficheiro com o código necessário para a inicialização/arranque da aplicação.

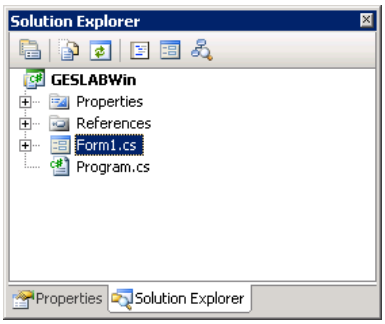


**Figura 2 – Exemplo de um *Solution Explorer***

## Program.cs

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;

namespace WindowsApplication1
{
   static class Program
   {
      /// <summary>
      /// The main entry point for the application.
      /// </summary>
      [STAThread]
      static void Main()
      {
         Application.EnableVisualStyles();
         Application.SetCompatibleTextRenderingDefault(false);
         Application.Run(new Form1());
      }
   }
}
```

Por vezes, pode suceder que a definição da localização de armazenamento (Passo 4) não esteja disponível, devido à opção **Save new projects when created** estar desactivada em **Tools | Options| Projects and Solutions | General**. Torna-se assim necessário, após a criação da solução/projecto inicial, definir a localização seleccionando: **File | Save *<nomedasolução>***, sendo apresentada a janela **Save Project** (Figura 3).

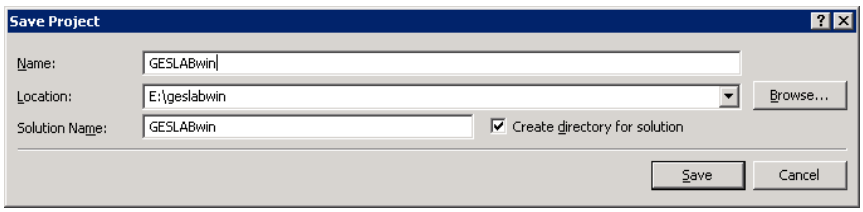


**Figura 3 – Exemplo da caixa de diálogo *Save Project***

## 2.2 O IDE do *Visual Studio*

Após a criação de um projecto aparece o IDE do *Visual Studio* no modo *Design View* (Figura 4), também conhecido por *Windows Form Designer*, para desenvolvimento da solução/projecto. Este IDE permite ao programador ter num único espaço funcionalidades que lhe permitem gerir os ficheiros do projecto (**Solution Explorer**), acesso aos controlos (**Toolbox**) para adicionar no formulário e alteração das propriedades dos controlos (**Properties**). Para adicionar controlos ao formulário, é usada a técnica "*drag and drop"*, da *toolbox* para o formulário (*Form*).

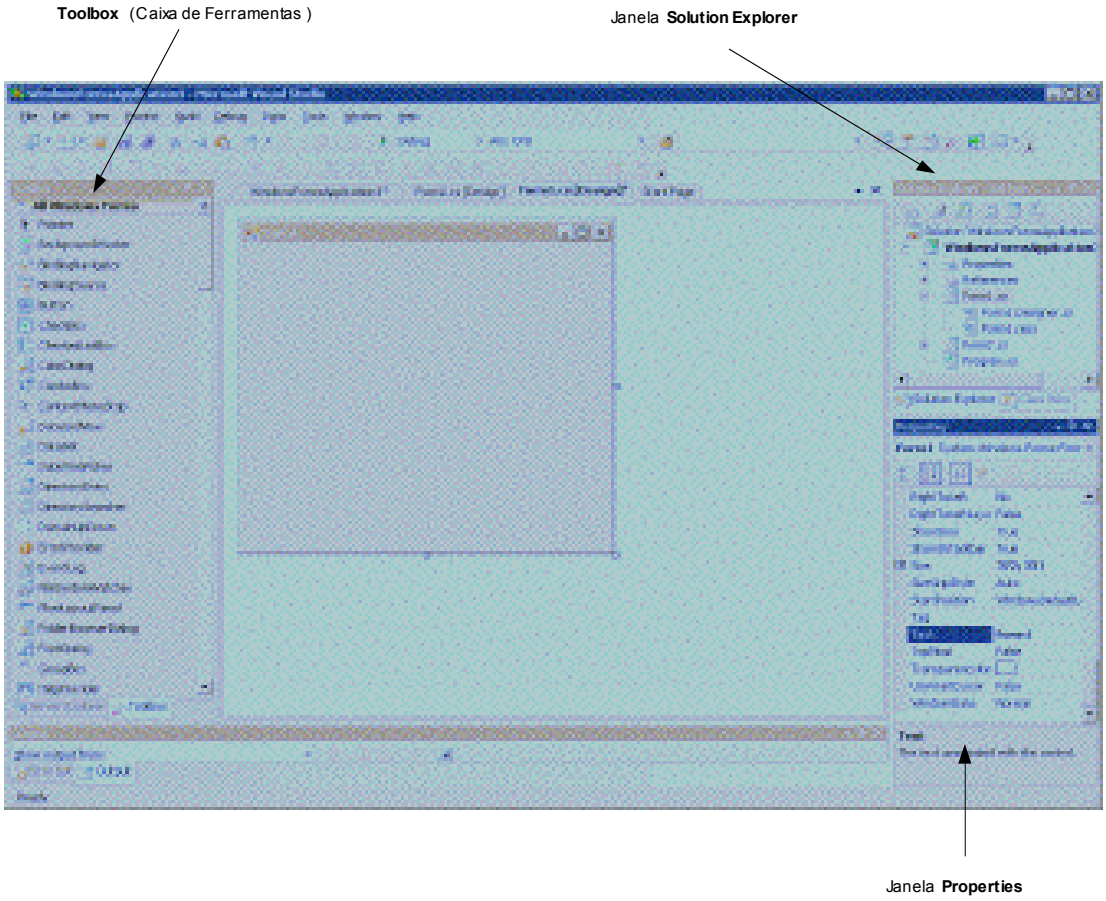


**Figura 4 – Exemplo do ambiente de trabalho do *Visual Studio***

De referir que o IDE pode ser organizado ao gosto de cada um, através do arrasto de cada uma das janelas para o local pretendido.

# 3 Programas simples

## 3.1 Mensagem de texto: “Ola Mundo!!!”

**Explicação**

Uma simples mensagem de texto pode ser apresentada de modo simples dentro de uma janela , através da classe **MessageBox**.

**Exemplo**

No Visual Studio deve criar-se um novo projecto, do tipo *Windows Application* (versão 2005) *ou Windows Forms Application* (versão 2008). De modo automático, o programa (IDE) cria uma janela (*Form*) e código mínimo em C#. Basta seleccionar a janela **Solution Explorer** e escolher o ficheiro (ou seja, clicar com o rato) chamado **Program.cs** (ou similar). Todo o seguinte código é gerado automaticamente pelo IDE, excepto a parte a **negrito**, que se irá alterar para este exemplo específico. O comentário (//) impede que a janela principal seja mostrada no ecrã. Eis o código:

**Program.cs**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;
namespace WindowsFormsApplication1
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            //Application.Run(new Form1());
            MessageBox.Show("Ola Mundo!!!");
        }
    }
}
```

**Resultado**


**Figura 5 – Janela da mensagem "Olá Mundo!!!"**

## 3.2 Resposta a uma pergunta de uma *MessageBox*

**Explicação**

Na maioria das situações, pretende-se utilizar a janela de mensagem de forma mais interactiva do que a apresentada no exemplo anterior, adicionando: *a)* um título à janela, *b)* interacção com o utilizador, e *c)* um ícon ilustrativo da mensagem/questão.

Uma **MessageBox** pode contar mais botões além do **OK**, permitindo desta forma recolher a resposta do utilizador a uma questão apresentada na janela de mensagem. Embora possam ser incluídos até três botões numa **MessageBox**, a sua disposição não pode ser definida. Ao programador é apenas permitido seleccionar um dos grupos pré-definidos de botões disponíveis (ver Tabela 1).

**Tabela 1 – Os botões de uma *MessageBox***

| **Valor** | **Botões** |
|---|---|
| *AbortRetryIgnore* | Abort, Retry, Ignore |
| *OK* | OK |
| *OKCancel* | OK, Cancel |
| *RetryCancel* | Retry, Cancel |
| *YesNo* | Yes, No |
| *YesNoCancel* | Yes, No, Cancel |

Em algumas situações, é usual surgir na janela da mensagem um ícone, normalmente ilustrativo da mensagem ou questão apresentada, estando disponíveis 9 possibilidades (Tabela 2):

**Tabela 2– Ícones para a *MessageBox***

| Valor | Descrição |
|---|---|
| *Asterisk* | Mostra um círculo contendo um *i* |
| *Error* | Mostra um círculo vermelho contendo um *X* |
| *Exclamation* | Mostra um triângulo amarelo com um ponto de exclamação. |
| *Hand* | Mostra um círculo vermelho contendo um X branco |
| *Information* | Mostra um círculo contendo um i |
| *None* | Não é mostrado qualquer icon |
| *Question* | Mostra um círculo contendo um ponto de interrogação |
| *Stop* | Mostra um círculo vermelho contendo um X branco |
| *Warning* | Mostra um triângulo amarelo contendo um ponto de exclamação |

### Exemplo

Vamos criar uma **Windows Application** que mostra uma janela de mensagem com três opções: "Sim", "Não" ou "Cancelar", quando a aplicação inicia (Figura 6). A resposta do utilizador é armazenada e apresentada uma segunda janela de mensagem com a opção escolhida pelo utilizador.

**Program.cs**

```
static void Main()
        {
            int resposta;
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);

           //Armazenamento da resposta do utilizador na MessageBox
           //Botão Yes - Valor 6
           //Botão No - Valor 7
           //Botão Cancelar - Valor 2
           resposta = Convert.ToInt32(MessageBox.Show("Pretende continuar?",
"Mensagem", MessageBoxButtons.YesNoCancel, MessageBoxIcon.Question));

        //Avaliação da resposta dada
        if (resposta == 6)
            MessageBox.Show("Carregou em Sim", "Mensagem", MessageBoxButtons.OK,
MessageBoxIcon.Information);
```

```
        else
            if (resposta == 7)
            MessageBox.Show("Carregou em Não", "Mensagem", MessageBoxButtons.OK,
MessageBoxIcon.Information);
        else
            if (resposta == 2)
            MessageBox.Show("Carregou em Cancelar", "Mensagem",
MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
```

---

**Resultado**

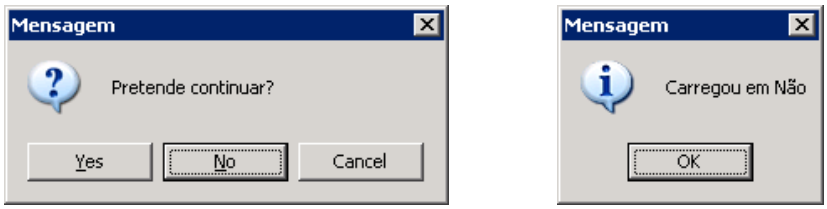


**Figura 6 - Janela de mensagem com botões de interacção**

## 3.3 Criação de uma janela (*Form*)

**Explicação**

Uma *Windows Aplication* baseia-se no princípio do ambiente *Windows*, ou seja, é baseada em janelas, sendo um tipo de aplicação em que a interacção com o utilizador é realizada através de formulários (*Form*). A janela principal corresponde à *Form* que é activada de início, normalmente através da instrução automática **`Application.Run(new Form1());`** da função **Main**. Uma *Form* tem diversas propriedades (título, cor, tamanho, etc..) que podem ser alteradas ao gosto do programador na janela de propriedades, quando o *Form* está seleccionado. As propriedades disponíveis na janela de propriedades (Figura 7) dependem do controlo seleccionado (e.g. **Form**, **TextBox**, **ListBox**).

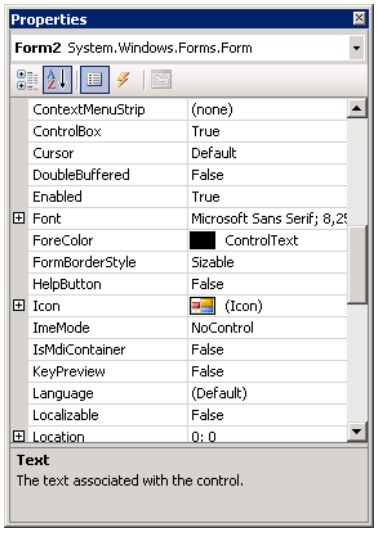


**Figura 7 – Exemplo da caixa de diálogo *Properties***

### Exemplo

Repetir o processo do exemplo anterior, ou seja, criar um nova aplicação *Windows*. Na opção **View** do IDE, escolher a visualização da janela **Properties**. Altera-se a propriedade "Size" para 200,100 e propriedade "Text" para "Janela Principal".

### Resultado

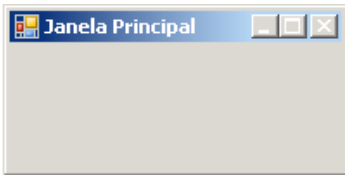


**Figura 8 – A janela principal da aplicação**

## 3.4 Criação de uma janela (Form) com um botão

### Explicação

Uma janela sem componentes é de utilidade reduzida. No geral, espera-se sempre alguma interactividade com o utilizador. Assim, é possível (e mesmo desejável) adicionar diversos componentes a uma janela. Cada componente tem as suas propriedades e eventos. É nestes últimos métodos que se adiciona código que irá fazer o programa alterar o seu comportamento, conforme a interacção com o utilizador. Neste exemplo irá testar-se uma interactividade com um botão simples.

**Exemplo**

Repetir o processo anterior, com a propriedade "Size" de 300, 300. Na opção **View** do IDE, escolher a visualização da janela **Toolbox**. Esta contém diversos componentes que podem ser inseridos numa *Form*. Arrastar o componente do tipo "button", e mudar a sua propriedade "Text" para "Mensagem". Com o rato posicionado dentro da *Form*, clicar 2 vezes no botão. Imediatamente, o IDE salta para o método **`button1_Click`** que activa o evento do botão. Acrescentar a linha a **negrito** e correr o programa:

```
private void button1_Click(object sender, EventArgs e)
        {
            MessageBox.Show("Ola Mundo!!!");
        }
```

**Resultado**

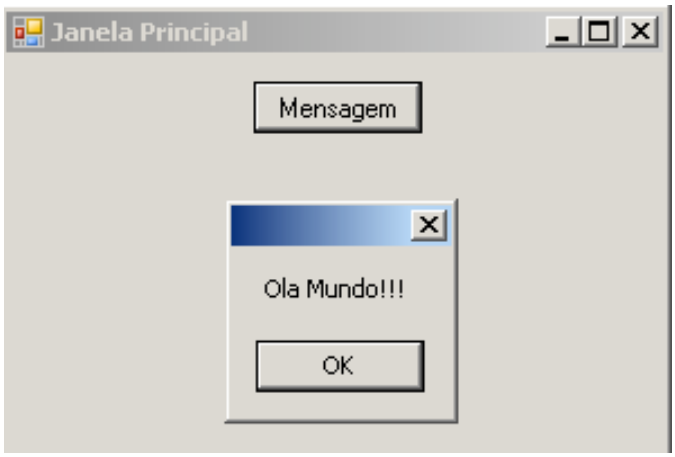


**Figura 9 – Caixa de mensagem activada por um botão**

## 3.5 Listagem de funcionalidades de alguns componentes

**Explicação**

O IDE disponibiliza um grande conjunto de componentes, estando fora do âmbito deste texto descrevê-los a todos. Os componentes servem para diferentes fins, sendo que de seguida se listam as suas principais funcionalidades:

- Mostrar texto: **Label**, **TextBox**, **LinkLabel**;

- Mostrar blocos : **Panel** (com ou sem bordo, com cores, etc), **GroupBox** (agrega componentes num quadrado);
- Áreas redimensionáveis: **Splitter**, **SplitContainer;**
- Mostrar Figuras: **PictureBox;**
- Para escolher uma ou mais opções: **Button** (uma), **CheckBox, CheckListBox** (uma ou mais), **RadioButton**, **ComboBox**, **ListBox**, **ListView** (somente uma de);
- Para escolher um valor variável: **TrackBar**, **HScrollBar**, **VScrollBar**;
- Para introduzir texto: **TextBox**;
- É possível ter tabulações **(TabControl)** e ménus dentro de um *Form* **(MenuStrip);**

# 4 Programas visuais que interagem com dados

O desenvolvimento de aplicações no *Visual Studio .Net*, seja qual for a linguagem de programação seleccionada (e.g. *Visual Basic*, C#), segue uma abordagem **Orientada aos Objectos**. As classes da aplicação podem ser geradas automaticamente, a partir da criação do **Diagrama de Classes** (*Class Diagram*) como item do projecto (Figura 10), sendo depois necessário proceder à codificação dos métodos.

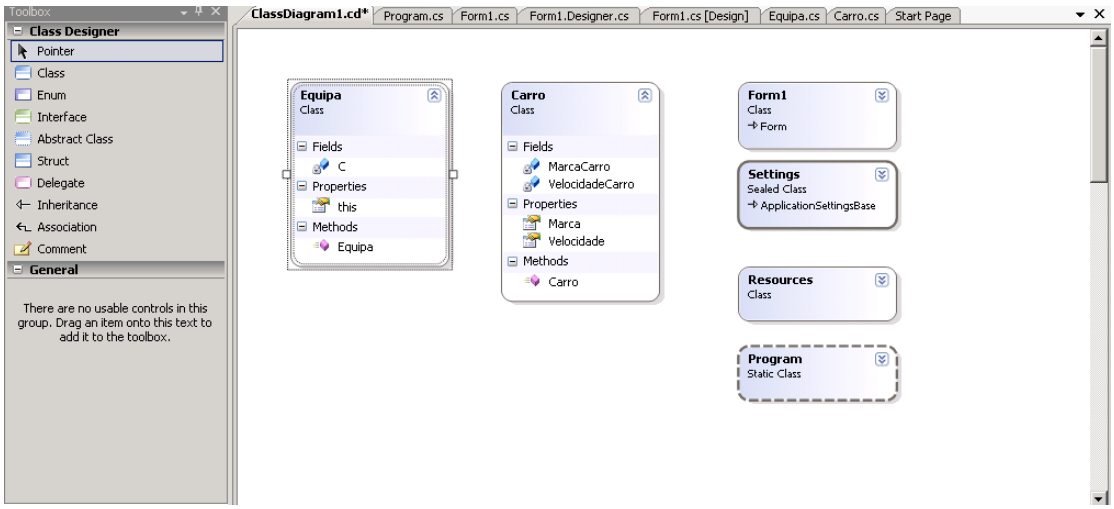


**Figura 10 – Exemplo da criação de um diagrama de classes**

Nesta secção, todos os projectos a criar são do tipo *Windows Application*, assegurando a interacção com o utilizador através de *Forms*.

## 4.1 Mudar os dados de um empregado

### Explicação

De modo geral, qualquer aplicação tem uma estrutura de dados (classes), sendo que o programa visual deve iteragir com esta estrutura de dados. Uma forma simples de realizar isto é inserir, como atributos privados, instâncias de classes (os objectos) dentro da *Form* principal. Embora o modo de programação dependa do programador, em geral os seguintes passos devem ser seguidos:

1. Criar a estrutura de dados (classes);

2. Desenhar a interface gráfica (*Form*) para interagir com esta estrutura de dados;
3. Incluir os objectos (instâncias de classes) como atributos privados da *Form*;
4. Programar os eventos para permitir a interação com os dados;

**Exemplo**

A classe Empregado é definida pelo seu nome e idade, sendo utilizados propriedades públicas para facilitar a manipulação das mesmas:

**Empregado.cs**

```
class Empregado
    {
        private string NomeEmp;
        private int IdadeEmp;
        public int Idade
        {
            get { return IdadeEmp; }
            set { IdadeEmp = value; }
        }
        public string Nome
        {
            get { return NomeEmp; }
            set { NomeEmp = value; }
        }
    }
```

Esta classe deve ser inserida dentro do projecto, por exemplo imediatamente antes da classe `Program`. De seguida, desenhar a interface gráfica. Para este exemplo escolheram-se os componentes:

- **Label** - para mostrar o texto, neste caso as perguntas;
- **TextBox** – para alterar o nome do empregado;
- **TrackBar** – para alterar a idade do empregado;
- **Button** – para actualizar os dados (idade);
- **GroupBox** – para separar de modo claro entre qual a parte que efectua a introdução de dados e a que realiza a visualização dos mesmos;

De seguida, mostra-se o aspecto da *Form* principal (Figura 11):

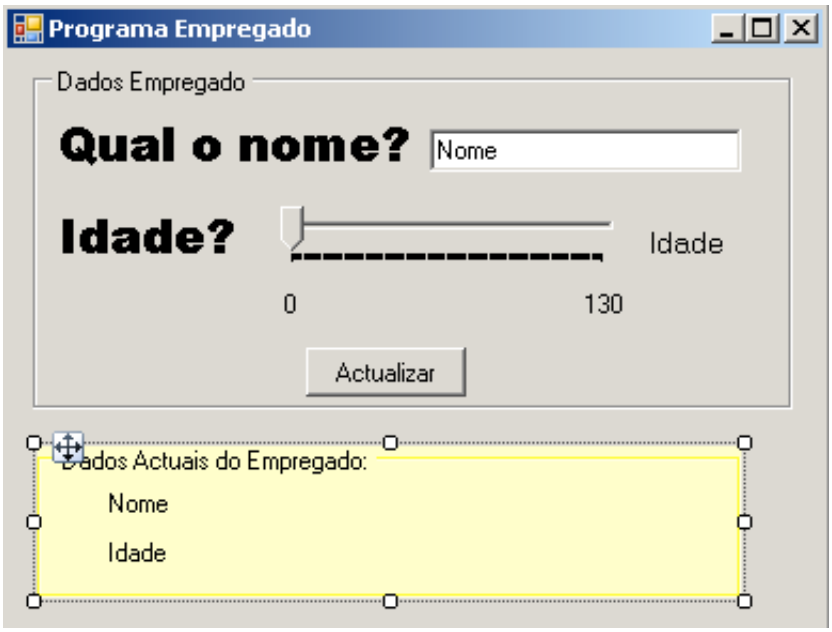


**Figura 11 – Formulário do programa empregado em modo *Design***

Dentro do código da janela principal (**Form1.cs**), acrescentou-se o atributo privado **E** e acrescentou-se código nos eventos dos componentes **trackBar1**, **textBox1** e **button1** (a negrito). O restante código foi gerado automaticamente pelo IDE:

**Form1.cs**

```
using System.Drawing;
using System.Text;
using System.Windows.Forms;
namespace Empregado
{
    public partial class Form1 : Form
    {
        private Empregado E;
        public Form1()
        {
            InitializeComponent();
            E = new Empregado();
        }
        private void button2_Click(object sender, EventArgs e)
        {
            E.Nome = textBox1.Text;
            label5.Text = E.Nome; label6.Text = (E.Idade).ToString();
        }
        private void trackBar1_Scroll(object sender, EventArgs e)
        {
            label7.Text = (trackBar1.Value).ToString();
            E.Idade = trackBar1.Value;
        }
        private void textBox1_TextChanged(object sender, EventArgs e)
        {
            E.Nome = textBox1.Text; label5.Text = E.Nome;
        }
```

```
    }
}
```

**Resultado**

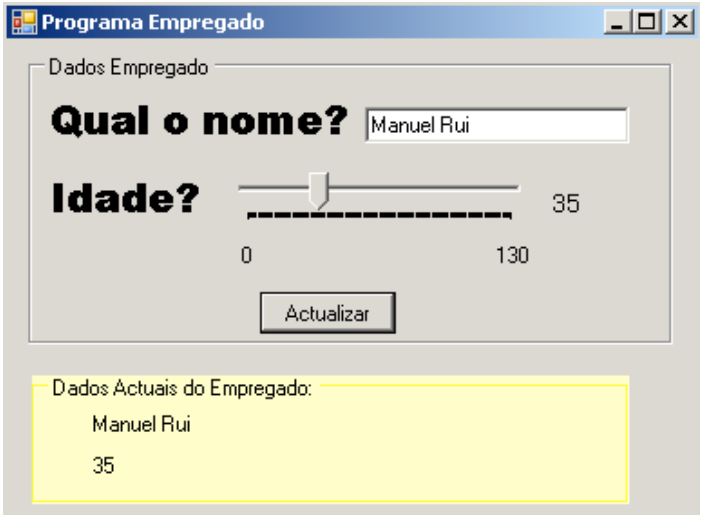


**Figura 12 – Formulário do programa empregado em modo *runtime***

## 4.2 Equipa de carros de corrida

**Explicação**

Uma classe pode utilizar objectos de outras classes como membros. Esta capacidade é chamada de **composição**, e é um dos princípios subjacentes à programação orientada a objectos no que concerne à reutilização de classes.

**Exemplo**

Neste exemplo vamos criar uma aplicação do tipo **Windows Application** para gestão/manipulação de 2 classes, em que uma das classes utiliza objectos da outra, seguindo o modelo de classes (*Class Diagram*) apresentado na Figura 13.

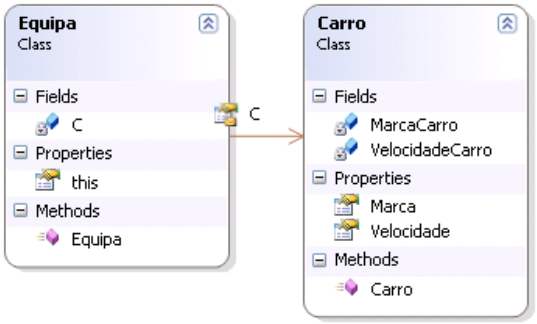


**Figura 13 – Diagrama de Classes com composição**

O programa deve permitir inserir os dados de cada carro, bem como alterá-los e mostrá-los no ecrã. Deve ser também mostrada a velocidade média de todos os carros.

1. Existe uma classe **Carro**, com uma velocidade actual (em km/h) e um nome de uma marca:

**Carro.cs**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace WindowsFormsApplication1
{
    class Carro
    {
        private string MarcaCarro;
        private int VelocidadeCarro;

        public Carro(string MarcaCarro, int VelocidadeCarro)
        {
            this.MarcaCarro = MarcaCarro;
            this.VelocidadeCarro =VelocidadeCarro;
        }


        public string Marca
        {
            get { return MarcaCarro ; }
            set { MarcaCarro  = value; }
        }
        public int Velocidade
        {
            get { return VelocidadeCarro; }
            set { VelocidadeCarro = value; }
        }
    }
}
```

2. Existe uma classe **Equipa** que contém *n* carros, cujo número de carros é passado ao construtor.

**Equipa.cs**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace WindowsFormsApplication1
{
    class Equipa
    {
        private Carro[] C;
        public Equipa(int n)
        {
            C = new Carro[n];
            for (int i = 0; i < n; i++)
                C[i] = new Carro("", 0);
        }

        public Carro this[int index]
        {
            get { return C[index]; }
            set { C[index] = value; }
        }

    }
}
```

3. Existe um formulário (Figura 14) para introdução e alteração dos carros da equipa, constituído pelos controlos da imagem.

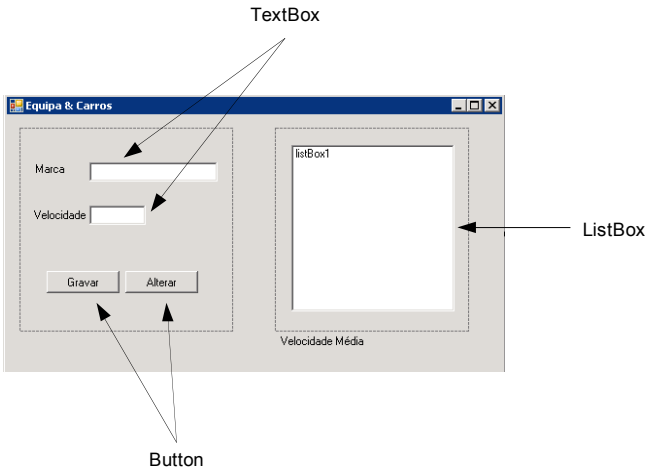


**Figura 14 – Formulário do projecto carros de corrida (*Design*)**

A **ListBox** tem como funcionalidade mostrar os carros da equipa, e seleccionar o carro a alterar. Quando o utilizador clica num dos carros exibidos na **ListBox**, os dados desse carro são apresentados nas **TextBox** respectivas permitindo a sua edição/alteração, que é confirmada no botão "Alterar".

As variáveis do formulário são:

*i* – índice para o array de carros;
*pos* – índice do elemento seleccionado na ListBox;
*E* – objecto do tipo Equipa.

Quando o utilizador introduz os dados de um novo carro para a equipa e clica no botão *Gravar*, é executado o método **`button1_Click`**, que resulta da associação do evento *click* no **button1**. Este método:

1. cria um novo objecto do tipo carro que é referenciado pelo objecto E do tipo Equipa (sendo necessário passar a indicação de qual a posição desse carro no vector – *i*);
2. incrementa o índice por forma a controlar o número de objectos do array;
3. acrescenta o objecto na **ListBox** (realizado através do método **`carrega_list()`**;
4. calcula e exibi a velocidade média dos carros da equipa (realizado através do método **`calcula_media()`**;.

Quando o utilizador selecciona um dos carros da equipa listados na **ListBox** os seus dados (marca e velocidade) são colocados nas caixas de texto e guardada na variável `pos` a posição no vector (necessária para saber qual a posição do objecto no vector de carros da equipa), sendo possível alterar os seus valores.
Para alterar os dados, é necessário clicar no botão "Alterar", que despoleta a execução do método **`button2_Click`**. Este por sua vez altera os dados do objecto, refrescando os dados na **ListBox** e recalculando a média.

**Form1.cs**

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        int i,pos;
        Equipa E = new Equipa(3);

        public Form1()
        {
            InitializeComponent();

        }

        private void Form1_Load(object sender, EventArgs e)
        {
            i = 0;

        }

        private void button1_Click(object sender, EventArgs e)
        {

            E[i] = new Carro(textBox1.Text,
Convert.ToInt16(textBox2.Text));
            i++;

            carrega_list();

            if (i>2)
                button1.Enabled = false;



            lblveloc.Text = Convert.ToString(calcula_media());
        }



        private void listBox1_SelectedIndexChanged(object
sender, EventArgs e)
        {

            pos =
Convert.ToInt16(listBox1.SelectedIndex.ToString());
```

```
            textBox1.Text="";
            textBox2.Text = "";
            textBox1.Text = E[pos].Marca;
            textBox2.Text=Convert.ToString(E[pos].Velocidade);



        }

        private void button2_Click(object sender, EventArgs e)
        {
            E[pos].Marca = textBox1.Text;
            E[pos].Velocidade=Convert.ToInt16(textBox2.Text);
            carrega_list();
            lblveloc.Text = Convert.ToString(calcula_media());

        }

        private void carrega_list()
        {
            int j;
            listBox1.Items.Clear();
            for (j=0;j<i;j++)
                listBox1.Items.Add(E[j].Marca + " - " +
E[j].Velocidade);

        }


        private double calcula_media()
        {
            int j;
            double media, soma;
            soma = 0;

            for (j = 0; j < i; j++)
            {
                soma = soma + E[j].Velocidade;
            }

            return (media = soma / j);

        }


    }
}
```

# 5 Soluções para janelas dentro de janelas

## 5.1. Como abrir uma nova *Form* dentro de uma *Form*?

**Explicação**

A classe *Forms* possui um método **`ShowDialog`** que mostra a *Form*.

**Exemplo**

Desenha-se a *Form* principal, sendo que lá dentro tem de existir um componente (ex. botão) que activa a nova *Form*. Cria-se uma nova *Form* (por exemplo Form2). Dentro do componente da *Form* principal, tem de existir um evento (ex. click_button1) com o seguinte código:

```
Form2 NN = new Form2(); // cria uma instancia da nova form
NN.ShowDialog();
```

## 5.2. Como impedir que múltiplas janelas apareçam fora da janela principal?

**Explicação**

A evolução em termos de complexidade das aplicações traduziu-se também no aumento do número de formulários necessários para o seu funcionamento, e no desenvolvimento do conceito de **MDI – Multiple Document Interface** [Patel, 2008]. Trata-se de um interface de programação *Windows* para criação de uma aplicação que permite aos utilizadores trabalharem com várias janelas ao mesmo tempo, com apenas um ícone na barra de tarefas. O formulário definido como MDI será o formulário "pai" da aplicação.

**Exemplo**

Pretende-se criar uma aplicação que contém um formulário principal ("pai") MDI, e um formulário "filho". No formulário principal será incluído um menu com a opção que permitirá abrir o formulário "filho".

Para criar a aplicação segundo os requisitos enunciados devem ser executados os seguintes passos:

1. Criar uma aplicação *Windows Application*;
2. Adicionar dois formulários: *Form1* e *Form2*.
3. Definir o *Form1* como formulário principal; para efectuar esta operação é necessário nas propriedades do formulário colocar a propriedade **IsMDIContainer** com o valor **True**, e opcionalmente a propriedade **WindowState** com o valor **Maximized**, para assegurar que a aplicação será aberta em modo maximizado.
4. Adicionar ao ***Form1*** o controlo **MainMenu** a partir da ToolBox (Figura 15).

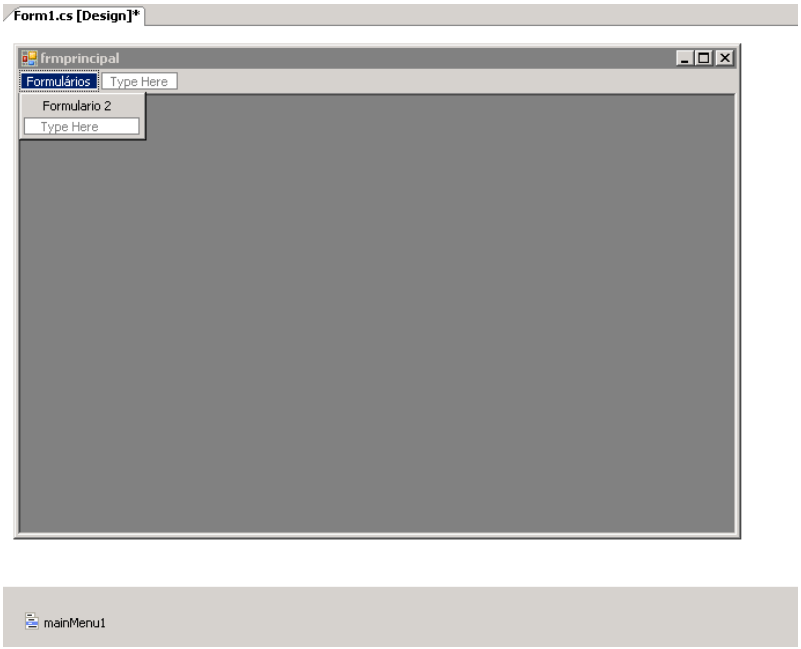


**Figura 15 – *MDI Form* com controlo *mainMenu***

5. Para abrir o *Form2* como "filho" do *Form1* é necessário que quando o método associado à opção do menu esteja codificado da seguinte forma:

```
private void menuItem2_Click(object sender, EventArgs e)
        {
           Form2 frm2;
           frm2 = new Form2();
           frm2.MdiParent = this;
           frm2.Show();
        }
```

A linha **frm2.MdiParent=this**, indica que a instância do formulário criada é filha da instância do *Form1* activa, que é do tipo MDI.

**Resultado**

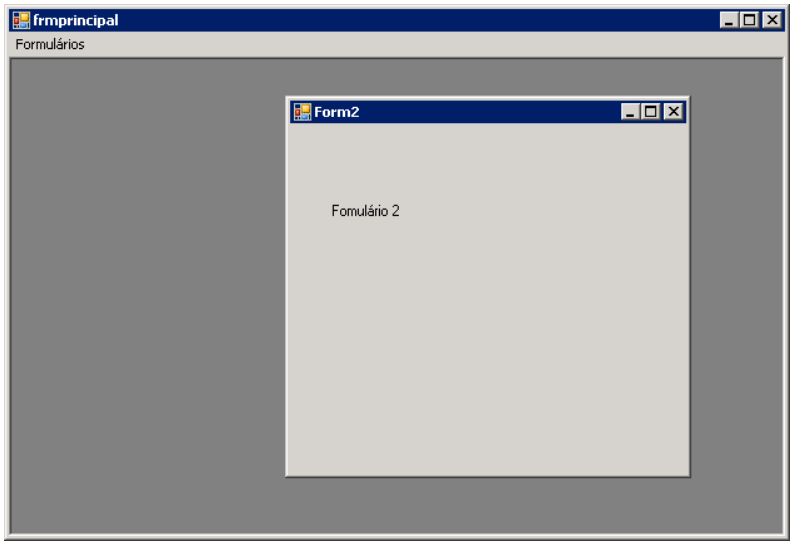


**Figura 16 – A aplicação MDI**

Apesar de estarem duas janelas activas, na barra de tarefas do *Windows* existirá apenas um ícone, correspondente ao formulário *"pai"*.

## 5.3 Como partilhar dados entre duas *Forms*?

**Explicação**

Em C# não existe o conceito de variáveis globais, como por exemplo em *Visual Basic,* onde podem ser utilizados os módulos. No entanto, isto pode ser um problema quando se pretendem partilhar dados entre formulários. Esta dificuldade pode ser ultrapassada pela criação de uma classe com a/as variáveis que se pretende utilizar como globais, declarando-as como estáticas (*static)*.

**Exemplo**

Pretende-se criar uma aplicação que contém a classe Aluno, um formulário principal ("pai") MDI, e dois formulários "filho" - *Form1* e *Form2*.

1. No *Form1* é criado um aluno.

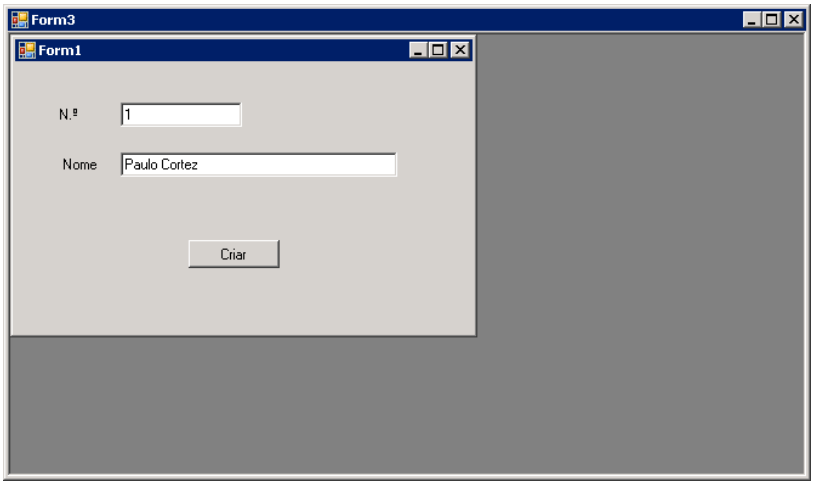


**Figura 17 – Formulário 1 de introdução de dados da aplicação *MDI***

2. No *Form2* os dados do aluno são alterados.

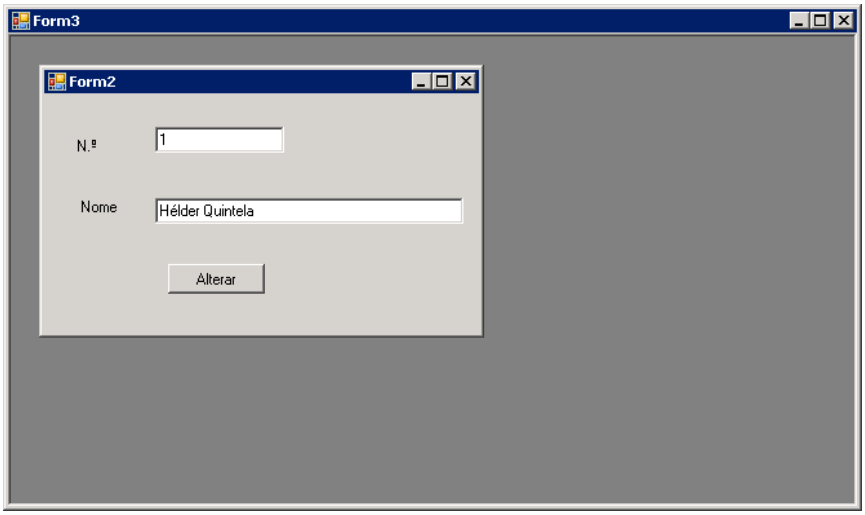


**Figura 18 - Formulário 2 de alteração de dados da aplicação *MDI***

3. Depois de alterados os dados do aluno no *Form2*, quando se regressa ao *Form1*, as caixas de texto apresentam os novos valores do objecto aluno que foi criado e posteriormente alterado.

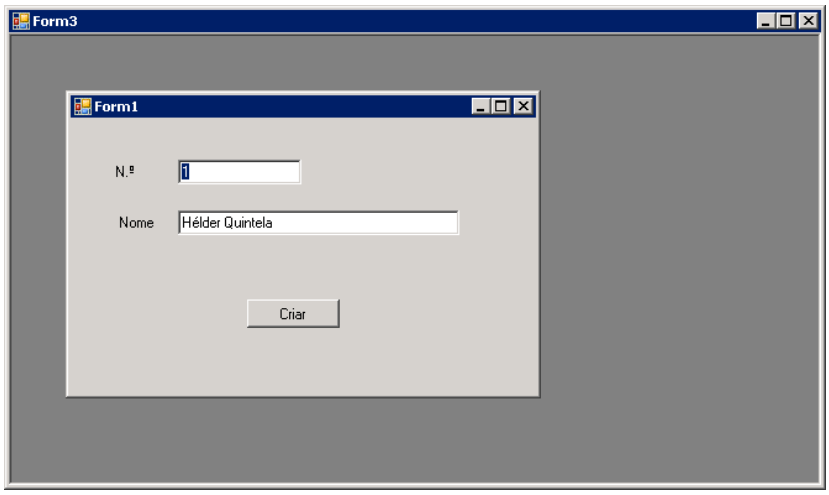


**Figura 19 - Formulário 1 actualizado da aplicação *MDI***

A classe **Aluno**, é composta por dois atributos: número e nome, estando implementados os métodos: construtor e acessores.

**Alunos.cs**

```
using System;
using System.Collections.Generic;
using System.Text;

namespace GlobalProject
{
    class Aluno
    {
        private int numero;
        private string nome;
        public Aluno(int numero, string nome)
        {
            this.numero = numero;
            this.nome = nome;
        }

        public int get_numero()
        {
            return numero;
        }

        public string get_nome()
        {
            return nome;
        }

        public void set_numero(int value)
        {
            this.numero = value;
        }

        public void set_nome(string value)
```

```
        {
            this.nome = value;
        }
    }

}
```

Para criar a variável "global", que será um objecto do tipo **Aluno**, cria-se uma classe com uma variável **a** static:

**Class1.cs**

```
using System;
using System.Collections.Generic;
using System.Text;

namespace GlobalProject
{
    class Class1
    {
        public static Aluno a;

    }
}
```

No *Form1* quando o é despoletado o evento *click* no botão "Criar", o valor da variável **a** da **Class1** será um novo objecto do tipo **Aluno**, sendo invocado o respectivo método construtor.

**Form1.cs**

```
...
private void button1_Click(object sender, EventArgs e)
        {
            Class1.a = new Aluno(Convert.ToInt32(textBox1.Text),
textBox2.Text);
            Form2 mform2 = new Form2();
            mform2.MdiParent = Form3.ActiveForm;
            mform2.Show();
            this.Close();
        }
...
```

No *Form2* quando o é despoletado o evento click no botão "Alterar", os valores dos atributos do objecto são alterados, sendo para o efeito invocados os métodos acessores da classe **Aluno**.

**Form2.cs**

```
...
private void button1_Click(object sender, EventArgs e)
        {
            Class1.a.set_numero(Convert.ToInt32(textBox1.Text));
            Class1.a.set_nome(textBox2.Text);
            Form1 frm1 = new Form1();
            frm1.MdiParent = Form3.ActiveForm;
            frm1.Show();
                this.Close();

             }
...
```

# Bibliografia

**[Marques e Pedroso, 2005]** Marques, P. e Pedroso, H., C# 2.0, editora FCA.

**[Patel, 2008]** Patel, I., MDI (Multiple Document Interface), C# Corner, URL:*http://www.csharpcorner.com/UploadFile/IrfanPatel/MultipleDocumentInterface11232005034108 AM/MultipleDocumentInterface.aspx*, 2008.

## Bibliografia adicional (recomendada para saber mais):

**[Whatis?com, 2008]** Whatis?com, What is a Multiple Document Interface?, URL: *http://whatis.techtarget.com/definition/0,,sid9_gci214090,00.html*.

**[Jones and MacDonald, 2005]** Jones, A. and MacDonald, M., *C# 2005 Recipes – A Problem-Solution Approach*, Apress.

**[Deitel, 2005]** Deitel, H.M., *Visual C#2005: How to Program, Second Edition*, Prentice Hall.

**[Robinson et al., 2004]** Robinson, S., Nagel, C., Glynn, J., Skinner, M., Watson, K and Evjen B., *Professional C# Programmer To Programmer, 3rd Edition*, Wrox.

**[Marshall, 2006]** Marshall, D., *Programming Microsoft Visual C# 2005: The Language*, Microsoft Press.